Zoea

Research note 32

Developer Half-life and the Value of AI Generated Code

Edward McDaid & Sarah McDaid
2 Jun 2024

Developer half-life is the time required for 50% of a developer's code to be eroded by later commits. This measure correlates directly with developer experience and represents a proxy metric for code value. In a mixed human-AI codebase, half-life could be used to assess the relative value of AI generated code.

There have been many attempts over the years to somehow quantify the 'value' delivered by individual software developers and the use of such metrics is common, particularly in larger organisations. While the whole idea is thoroughly distasteful to many developers, everybody forms some opinion regarding the capability of their colleagues which can be positive or negative.

Traditional software development metrics involve direct measurements, like counting the number of lines of code produced. These approaches are crude at best and they are frequently criticised by developers who recognise that not all code is the same, valuable code is not necessarily large, more experienced developers tend to write more compact code and that any such approach can easily be gamed.

The perceived value of code depends on what perspective you are taking. The business will place a higher value on software that directly delivers revenue or reduces costs. Developers on the other hand have a greater awareness of code that makes development easier or more difficult.

As a result, measuring code value is more involved than simple metrics like size or complexity. In effect, we somehow have to determine how people perceive software. Static analysis of code is not sufficient to accomplish this.

Each piece of code has its own timeline that involves not just the original author but potentially many other members of the development team. As a result, the way in which the entire development team interacts with the code for a given developer might indicate something about how they regard that code.

In a revision controlled codebase - with many contributors - it is possible to determine who committed any particular piece of code, for example using the blame command in subversion. This allows us to identify - at any point in time - all of the code that was most recently touched by a given developer.

When a developer permanently leaves a team, the code they leave behind is gradually modified or disappears over time as a result of other peoples updates. This includes code being replaced, deleted and refactored. The length of time it takes for the amount of code attributed to a given coder to reduce by 50% is what we call the developer half-life. Typically this reduction follows an exponential decay pattern although abrupt changes and periods of no change can also occur. The use of the term half-life is analogous to the decay of a radioactive material.

We can also determine the half-life for developers that are current team members. At a given point in time the half-life is the length of time for half of the developers code that was in existence at that time to change or disappear - ignoring any additional code that they might have added later. By convention, code that is modified by the same developer is also treated as though it has lost its original attribution.

The half-lives for a number of developers on the same team often exhibit considerable variation. It is also apparent that there is a strong correlation between half-life and the level of experience of the individual. Here experience means length of time on the current team. Less experienced developers tend to have shorter half-lives - often measured in weeks or months - whereas more experienced developers will have much longer half-lives, measured in years.

Clearly, half-life depends on more than simply the level of experience of each developer. The size and composition of the team, its culture and dynamics are also important. As such the actual durations will vary from team to team. However, it is the relative measurements calibrated within a single team that are useful.

It is worth noting that the half-life for given developer changes gradually over time. In general, a developer's half-life increases as their experience grows, although this is not always the case.

Aside from being an interesting curiosity what does developer half-life really tell us? Fundamentally, this metric says something about the duration for which the code produced by a given developer persists in the codebase. It can be argued that code which avoids or survives the scrutiny of colleagues has higher level of value than that which was quickly modified or deleted. Half-life therefore provides a proxy measure for code value that is decoupled from the quantity of code produced. Half-life can indicate the real value of a developer's output - even when this is out of step with that individuals current role and grade.

The generation of software using AI is rapidly becoming more common. This means that we are entering an era in which codebases are likely to contain a mixture of human and computer originated code. Under these circumstances it would be helpful to quantify the added value that AI brings to the development team. Developer half-life can easily be calculated for AI produced code - assuming it is suitably attributed in the revision control system.

Currently, AIs generate code as a result of some human produced specification - text prompts in the case of large language models and test cases with Zoea. This suggests we should measure the half-life for generated code separately for each AI user. Therefore, some traceability between specification and generated code will also be required. Such specifications also need to be subject to version control and it wouldn't be surprising if they demonstrated the same sort of half-life characteristics.

It is possible that the half-lives for generated code will exhibit a range of values and they may also change over time. Different AI based coding tools may also result in different half-lives. In any event, it will be readily apparent how well the value of AI generated code stacks up against the human variety.

Learn more at **zoea.co.uk**